Tree Pattern Relaxation

Sihem Amer-Yahia
AT&T Labs–Research
http://www.research.att.com/~sihem/

Joint work with SungRan Cho and Divesh Srivastava (EDBT’02)
Outline of Talk

- XML preliminaries
- Why relaxations?
- Which relaxations?
- Threshold problem
- Experimental results
XML Example: Database and Query

A database instance

A tree pattern
Motivation: Why Approximate Matching?

- Traditional query semantics: exact matching
  - embedding query in database instance
- Motivation for approximate matching: XML data heterogeneity
  - schema might not be defined or known
  - schema often allows optional elements
  - too few or no exact matches
Motivation: Approximation via Relaxation

- Relaxation = single query modification
  - compute superset of exact matches
- Relaxed query $Q' = Q+$ composition of relaxations
- Approximate match to $Q = $ exact match to $Q'$
  - enumerate relaxed queries $Q'$ of $Q$
  - efficiently compute exact matches to each $Q'$
- Goal: return “best” approximate matches in ranked order
Which Query Relaxations?

- Node Generalization: node type $\rightarrow$ supertype
- Leaf Node Deletion: remove leaf node and edge to parent
- Edge Generalization: pc-edge $\rightarrow$ ad-edge
- Subtree Promotion: subtree $\rightarrow$ ad-edge with grandparent
Example: Relaxed Queries

- **Original Query**
  - Book
    - Collection
      - Name
    - Editor
      - Address

- **Node Generalization**
  - Document
    - Collection
      - Name
    - Editor
      - Address

- **Leaf Node Deletion**
  - Book
    - Collection
      - Name
    - Editor

- **Approximate Matches**
  - Edge Generalization
  - Subtree Promotion
  - Composition
Efficient Evaluation of Relaxed Queries

- Exact matching of original query: use a join plan
  - binary containment/structural joins
- Encode desired query relaxations into the join plan
  - node generalization: node predicate $\rightarrow$ supertype
  - leaf node deletion: innerjoin $\rightarrow$ outerjoin
  - edge generalization: join predicate $c(A, B) \rightarrow d(A, B)$
  - subtree promotion: modified join predicate
Example: Original Join Plan

A tree pattern

A query evaluation plan
Example: Encoding Node Generalization

A tree pattern

generalize node Book to Document

Document

Collection

Editor

Address

Name

c (Document, Editor)

c (Document, Collection)

Book

Collection

Editor

Name

Address

A tree pattern
Example: Encoding Leaf Node Deletion

A tree pattern

make node Address optional
Example: Encoding Edge Generalization

A tree pattern

relax edge Editor--Name

\[
c (\text{Editor, Name}) \\
\text{OR} \\
((\text{NOT} \text{ exists} (c (\text{Editor, Name}))) \text{ AND} d (\text{Editor, Name}))
\]
Example: Encoding Subtree Promotion

A tree pattern

promote subtree rooted at node Name

\[
\text{promote subtree rooted at node Name}
\]
Example: Encoding a Composition

```
<table>
<thead>
<tr>
<th>Document Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding all Tree Pattern Relaxations</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>d (Editor, Address) OR ((NOT exists (d (Editor, Address)) AND d (Document, Address))</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (Editor, Name) OR ((NOT exists (c (Editor, Name)) AND d (Editor, Name)) OR ((NOT exists (d (Editor, Name)) AND d (Document, Name))</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (Document, Editor) OR ((NOT exists (c (Document, Editor)) AND d (Document, Editor))</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>Document Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding all Tree Pattern Relaxations</td>
</tr>
</tbody>
</table>

```
Assigning Scores to Approximate Matches

- **Essential for ranking approximate matches**
  - score = measure of “closeness” to original query
- **Use weighted tree patterns**
  - (exact, relaxed) weights with nodes and edges
  - flexible mechanism to represent different preferences
- **Score of an approximate match**
  - sum of exact and relaxed contributions
  - relaxed contribution = $f$(relaxations encoded)
Example: Weighted Tree Pattern and Scores

A database instance

A weighted tree pattern

Maximum score = 45
Threshold Problem

• Threshold problem: Given a weighted tree pattern $Q$, and a threshold $t$, find all approximate matches to $Q$ whose score $\geq t$
  º one can also formulate a Top-K problem

• Evaluation problem: Given a join evaluation plan that encodes all relaxations, efficiently compute all matches whose score $\geq t$
  º Algorithm Thres: prune partial results early
  º Algorithm OptiThres: unencode irrelevant relaxations
Algorithm Thres

Algorithm Thres(Node n)
    if (n is leaf) {
        list = evaluateLeaf(n);
        for (r in list)
            if (r→score + n→maxW ≥ threshold) append r to results;
        return results;
    }
list1 = Thres(n→left); list2 = Thres(n→right);
for (r1 in list1) {
    for (r2 in list2) {
        if (checkPredicate(r1,r2,n→predicate))
            s = computeScore(r1,r2,n→predicate);
        if (s + n→maxW ≥ threshold)
            append (r1,r2) to results with score s; }
        if ( ∃ r2 that joins with r1)
            if (r1→score + n→maxW ≥ threshold)
                append (r1,-) to results with score r1→score;
    }
return results;

• maxW: how much can the score of a partial result increase?
Example: Weights Used by Algorithm Thres

A weighted tree pattern

maxW used by Algorithm Thres

Maximum score = 45
Algorithm OptiThres

Algorithm OptiThres(Node n)
    if (n is leaf) {
        // evaluate, prune, and return results as in Algorithm Thres
    }
    list1 = OptiThres(n→left);
    /* maxLeft is set to the maximal score of results in list1 */
    if (maxLeft + relaxNode < threshold) unrelax(n→right);
    list2 = OptiThres(n→right);
    /* maxRight is set to the maximal score of results in list2 */
    if (maxLeft + relaxJoin < threshold) unrelax(n→join);
    if (maxLeft + maxRight + relaxPred < threshold)
        unrelax(n→join→predicate);
    // now, evaluate, prune and return join (and possibly outer join)
    // results as in Algorithm Thres

- relaxNode: should right child of join node remain relaxed?
- relaxJoin: should join remain an outer join?
- relaxPred: should join predicate remain relaxed?
Example: Weights Used by Algorithm OptiThres

<table>
<thead>
<tr>
<th>Query pattern</th>
<th>Query evaluation plan</th>
<th>Actual OptiThres evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7, 2)</td>
<td>(?, 0, 2)</td>
<td>c (Document, Month)</td>
</tr>
<tr>
<td>Proceeding</td>
<td>OR</td>
<td>OR</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>d (Document, Month)</td>
<td>d (Document, Month)</td>
</tr>
<tr>
<td>Publisher</td>
<td>(10, 1)</td>
<td>c (Document, Person)</td>
</tr>
<tr>
<td>(6, 2)</td>
<td>OR</td>
<td>OR</td>
</tr>
<tr>
<td>Month</td>
<td>d (Document, Person)</td>
<td>d (Document, Person)</td>
</tr>
<tr>
<td>(2, 0)</td>
<td></td>
<td>Document Person</td>
</tr>
<tr>
<td>(7, 2)</td>
<td></td>
<td>Document Person</td>
</tr>
<tr>
<td>Proceeding</td>
<td></td>
<td>Publisher</td>
</tr>
<tr>
<td>(2, 1)</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>(6, 2)</td>
<td></td>
<td>Month</td>
</tr>
<tr>
<td>Publisher</td>
<td></td>
<td>c (Document, Publisher)</td>
</tr>
<tr>
<td>(10, 1)</td>
<td></td>
<td>OR</td>
</tr>
<tr>
<td>Month</td>
<td></td>
<td>d (Document, Publisher)</td>
</tr>
<tr>
<td>(2, 0)</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>(7, 2)</td>
<td></td>
<td>Publisher</td>
</tr>
<tr>
<td>Proceeding</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>(2, 1)</td>
<td></td>
<td>Month</td>
</tr>
<tr>
<td>(6, 2)</td>
<td></td>
<td>c (Document, Publisher)</td>
</tr>
<tr>
<td>Publisher</td>
<td></td>
<td>OR</td>
</tr>
<tr>
<td>(10, 1)</td>
<td></td>
<td>d (Document, Publisher)</td>
</tr>
<tr>
<td>Month</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>(2, 0)</td>
<td></td>
<td>Publisher</td>
</tr>
</tbody>
</table>

Sihem Amer-Yahia

Tree Pattern Relaxation
Experimental Evaluation: Setup

- DBLP dataset with 2.1 million elements
  - DTD and dataset: http://dblp.uni-trier.de/db

<table>
<thead>
<tr>
<th>Label</th>
<th>No. of elements</th>
<th>Label</th>
<th>No. of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>article</td>
<td>87,675</td>
<td>url</td>
<td>212,792</td>
</tr>
<tr>
<td>cdrom</td>
<td>13,052</td>
<td>ee</td>
<td>55,831</td>
</tr>
<tr>
<td>document</td>
<td>213,362</td>
<td>magazine</td>
<td>0</td>
</tr>
<tr>
<td>publisher</td>
<td>1,199</td>
<td>person</td>
<td>448,788</td>
</tr>
</tbody>
</table>

- dummy type magazine and supertypes document, person
Experimental Evaluation: Queries Used

Q1

\[
\begin{align*}
&\text{article} (5, 1) \\
&\text{url} (1, 0) \\
&\text{ee} (1, 0) \\
&\text{cdrom} (1, 0) \\
&\text{cdrom} \\
&\text{ee} \\
&\text{url} \\
&\text{article}
\end{align*}
\]

Q2

\[
\begin{align*}
&\text{magazine} (4, 2) \\
&\text{publisher} (4, 1) \\
&\text{document} \\
&\text{person}
\end{align*}
\]
Time Comparison: Thres vs postPrune

![Graph showing cumulative time comparison for different query evaluation steps and thresholds.](image)
Data Size Comparison: Thres vs postPrune

![Graph](image)

- "postPrune"
- "Threshold=2"
- "Threshold=3"
- "Threshold=5"
- "Threshold=8"

Query Evaluation Steps (Thres)

- Join(article, url)
- Join(article, ee)
- Join(article, cdrom)

Data Size
Time Comparison: OptiThres, Thres, postPrune

![Graph showing cumulative time for different query evaluation steps]

- "postPrune"
- "Thres"
- "OptiThres"

Cumulative Time (in seconds) vs. Query Evaluation Steps

- Join(magazine,publisher)
Data Size Comparison: OptiThres, Thres, postPrune

Query Evaluation Steps

Join(magazine,publisher)

Data Size

"postPrune"
"Thres"
"OptiThres"
Comparison with Rewriting-Based Approaches

<table>
<thead>
<tr>
<th>Evaluation Method</th>
<th>Evaluation Time</th>
<th>Number of Joins</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptiThres</td>
<td>18.550s</td>
<td>3</td>
</tr>
<tr>
<td>postPrune</td>
<td>22.384s</td>
<td>3</td>
</tr>
<tr>
<td>MultiQ</td>
<td>40.842s</td>
<td>10</td>
</tr>
<tr>
<td>MultiQOptim</td>
<td>30.782s</td>
<td>8</td>
</tr>
</tbody>
</table>

- Query Q1, with threshold set to 2
  - to select a large number of answers
Related Work

- Language proposals for approximate matching
  - XXL [TW00]: extends XML-QL for ranked retrieval
  - approXQL [Sch01]: extends XQL
- Specification and evaluation
  - relaxations in a mediator [DR01]: no evaluation
  - rewriting based on tree-edit distance [Sch01]: no weighting
  - flexible/semi-flexible [KS01]: no scoring, ranking
Conclusions

- Novel tree pattern relaxation techniques
  - encoding relaxations in join evaluation plans
  - substantial benefits over post-pruning and rewriting

- Open problems
  - analog of $tf \times idf$ used in Information Retrieval
  - integration with cost-based join ordering
  - estimation of number of answers for a given threshold